Лабораторна робота № 5. Перевантаження функцій і операцій.

Мета роботи: вивчення операторів перевантаження функцій і операцій, конструкторів копіювання, використання аргументів по замовчуванню.

Теоретичний матеріал лекції, [1, розділ 4], [2, розділ 14, 15], [3, розділи 8, 11].

1. Короткі теоретичні відомості

1.1. Перевантаження функцій

Перевантаження функцій – це використання одного імені для декількох функцій, які використовують різні типи або різну кількість параметрів.

```
int func(int i);
int func(int i, int j);
double func(double i);
```

Тип значення, яке повертає функція не дозволяє її перевантажувати

```
int func(int i);
double func(int i);
```

Іноді оголошення двох функцій зовнішньо відрізняється, але фактично співпадає

```
void func(int *p);
void func(int p[]);
```

1.2. Перевантаження операцій

Операції C++ є контекстнозалежними і можуть бути перевантажені у класі. Перевантажені операції використовуються як мікрооперації над об’єктами класів. Допускається перевантаження таких операцій:

- первинних

```
()      []      ->
```

-унарних

```
!       ~       +       -       ++      --      &       *       new     delete
```

- бінарних

```
->*     *       /       %       +       -       <<      >>
<       <=      >       >=      ==      !=      &       ^
|       &&      ||      =       *=      /=      %=      +=
-=      &=      ^=      |=      <<=     >>=     ,
```

Перевантаження операцій здійснюється за допомогою *операторних функцій*:

```
тип operator операція (список формальних аргументів) {…}
```

При перевантаженні не можна змінити пріорітети та асоціативність операцій.

Операторні функції можуть бути оголошені членами або друзями класу. Тільки членами класу оголошуються операції `()`, `[]`, `->`, `new`, `delete`, `=`. Усі інші операції можуть бути оголошені як членами, так і друзями класу. Операторні функції не можуть мати параметрів за замовчуванням.

1.3. Варіанти перевантаження операцій

Якщо операторна функція є членом класу, то її першим (неявним) параметром є вказівник `this`. Кількість параметрів операторної функції визначається її видом та способом оголошення - як члена або як друга класу, як подано в табл.4.1.

Таблиця 4.1

Варіанти перевантаження операцій

Операції	**Члени класу**	**Друзі класу**
Унарні	Без параметрів	Один параметр з типом класу, посиланням або вказівником на об’єкт класу

Бінарні	Один параметр (другий)	Два параметри (перший параметр повинен мати тип класу, посилання або вказівник на об’єкт класу)

1.4. Унарні операції члени і друзі класу

Унарна операція член класу перевантажується за допомогою нестатичної операторної функції без параметрів. Якщо унарна операція перевантажується як друг класу, то вона має один параметр з типом класу, посилання або вказівника на об’єкт класу.

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    A& operator-() const;
};
A& A::operator-() const
{ return A(-x); }

void main()
{
   A a(1), b;
   b=-a;
   //…
}
```

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    friend A operator-(const A&);
};
A operator-(const A& a)
{ return A(-a.x); }

void main()
{
   A a(1), b;
   b=-a;
   //…
}
```

1.5. Бінарні операції члени і друзі класу

Перевантаження бінарної операції як члена класу є нестатичною операторною функцією з одним параметром (другим, перший є вказівник `this`). Бінарна операція перевантажена як друг класу, є операторною функцією з двома параметрами (перший - змінна класу, посилання або вказівник на об’єкт класу).

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    A& operator-() const;
};
A& A::operator-(const A& a) const
{ return A(x-a.x); }

void main()
{
   A a(1), b(2), c;
   c=b-a;
   //…
}
```

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    friend  A  operator-(const  A&,
const A&);
};
A operator-(const A& b, const A& a)
{ return A(b.x-a.x); }

void main()
{
   A a(1), b(2), c;
   c=b-a;
   //…
}
```

1.6. Виклики операторних функцій

Можливі два варіанти викликів операторних функцій – явний та неявний. Явний виклик операторних функцій здійснюється так само, як і звичайних дружніх функцій або методів класу:

```
b=a.operator-()          b=-a       унарний мінус - метод класу
b=operator-(a)           b=-a       унарний мінус - друг класу
b=a.operator-(a)         c=b-a      бінарний мінус - метод класу
b=operator-(b,a)         c=b-a      бінарний мінус - друг класу
```

Неявний виклик операторних фінкцій будується на основі застосування відповідних

операцій до об’єктів:

```
b=-a;   // унарний мінус - метод класу
b=-a;   // унарний мінус - друг класу
c=b-a;  // бінарний мінус - метод класу
c=b-a;  // бінарний мінус - друг класу
```

1.7. Перевантаження первинних та унарних операцій

Операція `()` перевантажується як член класу, може повертати один з визначених типів та може мати змінну кількість аргументів.

Операція `[]` перевантажується як бінарна нестатична функція член класу. Якщо повертає посилання, то дозволяє використання з обох сторін операції присвоєння (функції з типом поилання можна присвоїти значення).

Операція `->` перевантажується як унарна постфіксна операція. Повинна бути нестатичною функцією членом і має повертати вказівник на структурований тип. Не повинна мати параметрів. Клас, який перевантажує операцію `operator->`, називають розумним (smart) вказівником.

Операції інкремента та декремента застосовуються у префіксній (`++x, --x`) та постфіксній (`++x, --x`) формах. Перевантажуються як унарні операції, можуть бути членами або друзями класу. Для розрізнення префіксної та постфіксної форм у постфікс ній формі використовують додатковий параметри типу `int`, який явно не задається:

- префіксна форма (метод класу – `A& operator++();` друг класу – `friend A& operator++(A&);`

- постфіксна форма (метод класу – `A& operator++(int);` друг класу – `friend A& operator++(A&, int)`.

Операції new та delete можуть бути перевантажені у контексті класу або без використання класу. У контексті класу є статичними, перевантажуються за допомогою методів, не можуть бути друзями, у них не можна використовувати нестатичні елементи класу, не можуть бути віртуальними. Операції `new` та `new[]` можуть мати більше одного параметра. Тоді вони оголошуються так:

```
void * operator new(size t, додатковий список параметрів);
void * operator new[](size t, додатковий список параметрів);
```

При виклику перевантажений операцій new, new[] додатковий список аргументів задається у круглих дужках після слова new.

```
A *p = new(додатковий список аргументів) A(аргументи конструтора);
A *q = new(додатковий список аргументів) A[кількість об’єктів];
```

Якщо додатковим параметром операції `new` є вказівник на раніше визначену змінну, то за допомогою такої операці можна розмістити об’єкт класу за відомою адресою. У цьому випадку динамічна пам’ять для об’єкта не виділяється, а операція `new` повертає адресу раніше виділеної області пам’яті.

1.8. Інші операції перевантаження

Операція присвоєння використовується для присвоєння одного об’єкту іншому існуючому об’єкту класу. Оголошується як нестатична функція член класу, не може бути константою, не успадковується, розміщується у відкритій частині класу.

Алгоритм перевантаження операції присвоєння подібний до алгоритму конструктора копіювання. Відмінність полягає у тому, що конструтор копіювання створює нову копію об’єкта, а в операції присвоєння використовуються раніше створені об’єкти. Якщо клас містить поля, які є вказівниками або посиланнями на будь-який тип, то операція присвоєння повинна бути перевантажена:

```
A& A::operator=(A& a) {
   if (this!=&a) { delete &r; r=*new int(a.r) }
```

```
    return this;
}
```

Операторна функція перетворення типу призначена для перетворення об'єкта класу у значення іншого типу, зокрема у тип іншого класу:

```
operator інший_тип();
```

Операція допускає перевантаження. У протоколі класу може бути оголошено декілька операторних функцій перетворення типу. Повертає значення, яке має "інший тип". Може бути віртуальною, успадковується.

Потокові операції введення-виведення перевантажуються як друзі класу, оскільки їх першим параметром повинен бути об'єкт потокового типу: `istream` – для введення, `ostream` – для виведення об'єктів класу. Перевантаження операцій потокового введення-виведення дає можливість цілісно виводити об'єкти оголошеного класу.

Запитання.

1. Що таке перевантаження функцій.
2. Як викликати через адресу перевантажену функції і передати їй параметр.
3. Аргументи за замовчуванню як альтернатива перевантаження функцій.
4. Неоднозначності, які виникають при перевантаженні функцій.
5. Що таке перевантаження операцій і який синтаксис операторної функції.
6. Особливості перевантаження унарних і бінарних операторних функцій, як членів класу і як друзів класу.
7. Перевантаження первинних операцій `()`, `[]`, `->`.
8. Перевантаження унарних операцій інкремента `++` та декремента `--`.
9. Перевантаження унарних операцій `new` та `delete`. Розміщувана форма операції `new`.
10. Перевантаження операції присвоєння =.
11. Операція перетворення типу `operator інший тип()`.
12. Перевантаження потокових операцій введення-виведення.

Завдання.

1. Створити класс ДІЙСНИХ ЧИСЕЛ. Перевантажити бінарні операції `+`, `-`, `*`, `/` для виконання арифметичних обчислень над дійсними числами. Операції `+`, `-` перевантажити за допомогою методів класу, а операції `*`, `/`, як дружні функції. Перевантажити потокову операцію >> для введення з клавіатури та операцію << для виведення на екран.

2. Оголосити клас ДОВГЕ ЦІЛЕ у двійковій системі числення. Двійкове число задати як рядок символів. Перевантажити арифметичні та логічні операції роботи з двійковими числами. Визначити операції введення з потоку та виведення у потік.

3. Оголосити клас МАТРИЦЯ. Перевантажити операції `+`, `-`, `*` відповідно для додавання, віднімання та множення матриць. Визначити операції введення матриці з потоку та виведення у потік.

4. Перевантажити операцію `[]` для визначення цілої, а операцію `()` – дробової частини дійсного числа. Визначити операцію введення з потоку та виведення у потік.

5. Перевантажити операцію `()` для виділення підмасиву із заданого масиву цілих чисел. Параметрами операції `()` є значення індексів елементів масиву. Визначити операції введення матриці з потоку та виведення у потік.

6. Перевантажити операції `->` та `[]` для звернення до елементів лінійного списку. Визначити операції введення з потоку та виведення у потік.

7. Перевантажити операції `new` та `delete` так, щоб перед знищенням об'єкта виводився час його існування у програмі. Визначити операції введення з потоку та виведення у потік.

8. Перевантажити операції `+`, `-`, `*` для виконання операцій над множинами цілих чисел. Визначити операції введення з потоку та виведення у потік.

9. Створити клас СТУДЕНТ, що має ім'я (вказівник на рядок), номер залікової книжки, оцінки з предметів. Визначити клас ГРУПА студентів, який містить список студентів. Перевантажити операції + для додавання нового елементу у список, – для вилучення зі списку даних про студента, який має незадовільні оцінки, * для об'єднання даних про студентів в один список. Визначити операції введення з потоку та виведення у потік.

10. Перевантажити операцію + для дописування одного файлу в кінець іншого, операцію – для вилучення з першого файлу елементів, які входять у другий файл, * для створення файлу, який складається зі спільних елементів обох файлів, ^ для встановлення позиції покажчика запису-читання файлу. Визначити операції введення з потоку та виведення у потік.

11. Створити клас КООРДИНАТА, який має дві координати `x` і `y`. Перевантажити операції `+, *` так, щоб можна було використовувати об'єкти типу `coord` для виконання інструкцій `ob+int, int+ob, ob*int, int*ob`.

12. Нехай задано наступне оголошення класу

```
class dynarray {
     int *p;
     int size;
public:
    dynarray(int s);   // передача розміру масива
    int &put(int i);   // повернення посилання на елемент i
    int  get(int i);   // повернення значення змінної i
    // створити функцію operator=()
};
```

Добавити все необхідне для створення типу динамічний масив. Тобто, виділити пам'ять для масиву і зберігти вказівник на цю пам'ять за адресою `p`. Розмір масиву в байтах зберегти в змінній `size`. Створити функцію `put()`, яка повертає посилання на заданий елемент масиву і функцію `get()`, яка повертає значення даного елемента. Забезпечити контроль границь масиву. Перевантажити інструкцію присвоєння так, щоб виділена кожному масиву такого типу пам'ять не була випадково пошкоджена при присвоєнні одного масиву іншому.

2. Завдання для самостійної роботи № 5

2.1. Перевантаження функцій

```
/* 1. розділ 14,listing 1 — перевантаження функцій з різним типом параметрів */
#include <iostream>
using namespace std;

int myfunc(int i); // ці варіанти відрізняються типами параметрів
double myfunc(double i);

int main()
{
  cout << myfunc(10) << " "; // виклик myfunc(int i)
  cout << myfunc(5.4);       // виклик myfunc(double i)
  return 0;
}

double myfunc(double i)
{
  return i;
}

int myfunc(int i)
{
  return i;
}
10 5.4

/* 2. listing 2 — перевантаження функцій з різною кількістю параметрів */
#include <iostream>
using namespace std;

int myfunc(int i); // ці варіанти відрізняються різною кількістю параметрів
int myfunc(int i, int j);

int main()
{
  cout << myfunc(10) << " "; // виклик myfunc(int i)
  cout << myfunc(4, 5);      // виклик myfunc(int i, int j)

  return 0;
}

int myfunc(int i)
{
  return i;
}

int myfunc(int i, int j)
{
  return i*j;
}
10 20


/* 3. listing 13 — визначення адресу перевантаженої функції */
/* Функція має адрес. Якщо цей адрес присвоїти вказівнику, то функцію можна
викликати не по імені, а через вказівник. Якщо функція перевантажена, то як
визначити її адрес? Адрес визначається по типу вказівника */
#include <iostream>
using namespace std;
```

```
int myfunc(int a);
int myfunc(int a, int b);

int main()
{
  int (*fp)(int a); // вказівник на функцію int f(int)
  //int (*fp) (int a, int b); // вказівник на функцію int f(int, int)
  fp = myfunc; // адрес функції на myfunc(int)
  cout << fp(5);
  return 0;
}

int myfunc(int a)
{
  return a;
}
int myfunc(int a, int b)
{
  return a*b;
}
```
5

/* 4. listing 18 — аргументам функції можна присвоїти значення по замовчуванню*/
```
#include <iostream>
using namespace std;
/* функція очистки екрана дисплея */
void clrscr(int size=25);

int main()
{
  register int i;

  for(i=0; i<30; i++ ) cout << i << endl;
  cin.get();
  clrscr(); // стирає 25 рядків

  for(i=0; i<30; i++ ) cout << i << endl;
  cin.get();
  clrscr(10); // стирає 10 рядків

  return 0;
}

void clrscr(int size)
{
   for(; size; size--) cout << endl;
}
```

/* 5. listing 20 — аргументи функцій по замовчуванню. Автоматична вставка перед рядком заданої кількості пропусків від краю екрана */
```
#include <iostream>
using namespace std;

/* За замовчуванням indent=-1 і функція використовує попереднє значення
відступів. */
void iputs(char *str, int indent = -1);

int main()
{
  iputs("Вітання всім", 10);
  iputs("Перед цим рядком вставлено 10 пропусків");
  iputs("Перед цим рядком вставлено 5 пропусків", 5);
  iputs("Перед цим рядком немає пропусків", 0);
```

```
  return 0;
}

void iputs(char *str, int indent)
{
  static int i = 0; // застосовується попереднє значення indent

  if(indent >= 0)
    i = indent;
  else              // використовується старе значення indent
    indent = i;

  for( ; indent; indent--) cout << " ";

  cout << str << "\n";
}
```

Вітання всім
Перед цим рядком вставлено 10 пропусків
Перед цим рядком вставлено 5 пропусків
Перед цим рядком немає пропусків

```
/* 6. listing 26 — аргументи по замовчуванню, як альтернатива перевантаженню */
#include <iostream>
#include <cstring>
using namespace std;

//void mystrcat(char *s1, char *s2);
//void mystrcat(char *s1, char *s2, int len);
/* функція еквівалентна двом попереднім, тобто реалізовує перевантаження */
void mystrcat(char *s1, char *s2, int len = -1);

int main() {
  char str1[80] = "This is a test";
  char str2[80] = "0123456789";

  mystrcat(str1, str2, 5); // конкатенація 5 символів
  cout << str1 << '\n';

  strcpy(str1, "This is a test"); // відновлення str1

  mystrcat(str1, str2); // конкатененація двох рядків
  cout << str1 << '\n';

  return 0;
}

// налаштована версія strcat()
void mystrcat(char *s1, char *s2, int len) {
  // знаходимо кінець рядка s1
  while(*s1) s1++;

  if(len == -1) len = strlen(s2);

  while(*s2 && len) {
    *s1 = *s2; // копіюємо символи
    s1++;
    s2++;
    len--;
  }

  *s1 = '\0'; // признак кінця рядка s1
}
```

```
This is a test01234
This is a test0123456789
```

```
/* 7. listing 28 — неоднозначність виклику перевантажених функцій */
#include <iostream>
using namespace std;

float myfunc(float i);
double myfunc(double i);

int main() {
  cout << myfunc(10.1) << " "; // однозначний виклик myfunc(double)
// неоднозначний виклик, неясно в який тип float чи double перетворити 10
  cout << myfunc(10);
  return 0;
}

float myfunc(float i) {
  return i;
}

double myfunc(double i) {
  return -i;
}
c14_28.cpp: In function ‘int main()’:
c14_28.cpp:11: error: call of overloaded ‘myfunc(int)’ is ambiguous
c14_28.cpp:5: note: candidates are: float myfunc(float)
c14_28.cpp:6: note:                 double myfunc(double)
```

```
/* 8. listing 29 — неоднозначний виклик функцій */
#include <iostream>
using namespace std;

char myfunc(unsigned char ch);
char myfunc(char ch);

int main()
{
  cout << myfunc('c');       // виклик функції myfunc(char)
  cout << myfunc(88) << " "; // неоднозначність => char чи unsigned char
  return 0;
}

char myfunc(unsigned char ch)
{
  return ch-1;
}

char myfunc(char ch)
{
  return ch+1;
}
c14_29.cpp: In function ‘int main()’:
c14_29.cpp:11: error: call of overloaded ‘myfunc(int)’ is ambiguous
c14_29.cpp:5: note: candidates are: char myfunc(unsigned char)
c14_29.cpp:6: note:                 char myfunc(char)
```

```
/* 9. listing 30  - неоднозначність, спричинена параметрами по замовчуванню */
#include <iostream>
using namespace std;

int myfunc(int i);
int myfunc(int i, int j=1);
```

```
int main()
{
  cout << myfunc(4, 5) << " "; // однозначність
  cout << myfunc(10);          // неоднозначність

  return 0;
}

int myfunc(int i)
{
  return i;
}

int myfunc(int i, int j)
{
  return i*j;
}
```

```
c14_30.cpp: In function ‘int main()’:
c14_30.cpp:11: error: call of overloaded ‘myfunc(int)’ is ambiguous
c14_30.cpp:5: note: candidates are: int myfunc(int)
c14_30.cpp:6: note:                 int myfunc(int, int)
```

```
/* 10. listing 31 — неоднозначність спричинена оператом &x */
/* В цій програмі помилка. Функція неможна перезавантажувати, якщо один параметр
передається по посиланню, а другий по значенню. */
#include <iostream>
using namespace std;

void f(int x);
void f(int &x); // error

int main()
{
  int a=10;
  f(a); // error, which f()?
  return 0;
}

void f(int x)
{
  cout << "In f(int)\n";
}

void f(int &x)
{
  cout << "In f(int &)\n";
}
```

```
c14_31.cpp: In function ‘int main()’:
c14_31.cpp:13: error: call of overloaded ‘f(int&)’ is ambiguous
c14_31.cpp:6: note: candidates are: void f(int)
c14_31.cpp:7: note:                 void f(int&)
```

2.2. Перевантаження операцій

```
/* 1. розділ 15, listing 1 — перевантаження оператора "+"  за допомогою операторної функції */
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;   /* клас містить географічні координати */
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";  // географ. довгота
    cout << latitude << "\n";  // географ. широта
  }

  loc operator+(loc op2);
};

// Перевантаження бінарного оператора + стосовно класу loc.
// лівий операнд this, а правий op2.
// При перевантаженні бінарного оператора виклик операторної
// функції генерується лівим операндом this
loc loc::operator+(loc op2)   // операторна функція
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30);

  ob1.show(); // виводить на екран 10 20
  ob2.show(); // виводить на екран 5 30

  ob1 = ob1 + ob2;
  ob1.show(); // виводить на екран 15 50

  return 0;
}
```

```
10 20
5 30
15 50
```

```
/* 2. listing 4 — перевантаження операторів "+", "-", "=", "++"  */
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
```

```
  loc() {} // конструктор для створення тимчасових об'єктів
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  loc operator+(loc op2);
  loc operator-(loc op2);
  loc operator=(loc op2);
  loc operator++();
};

// Перевантажений оператор "+" для класу loc.
loc loc::operator+(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}

// Перевантажений оператор "-" для класу loc.
loc loc::operator-(loc op2)
{
  loc temp;

  // Зверніть увагу на порядок елементів.
  // Дані об'єкта op2 мають відніматися від даних об'єкта, на які вказує
  // вказівник this
  temp.longitude = longitude - op2.longitude;
  temp.latitude  = latitude  - op2.latitude;

  return temp;
}

// Перевантажений оператор "=" для класу loc
loc loc::operator=(loc op2)
{
  longitude = op2.longitude;
  latitude = op2.latitude;

  return *this; // повернення об'єкта, який генерує виклик
}
// Перевантажений унарного оператор "++" для класу loc. Операнди функції
// передаються за допомогою вказівника this
loc loc::operator++()
{
  longitude++;
  latitude++;
  return *this;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30), ob3(90, 90);

  ob1.show();
```

```
  ob2.show();

  ++ob1;      // префіксний інкремент
  ob1.show(); // виводить на екран числа 11 21

  ob2 = ++ob1;
  ob1.show(); // виводить на екран числа 12 22
  ob2.show(); // виводить на екран числа 12 22

  ob1 = ob2 = ob3; // множинне присвоєння
  ob1.show(); // виводить на екран числа 90 90
  ob2.show(); // виводить на екран числа 90 90

  return 0;
}
```

10 20
5 30
11 21
12 22
12 22
90 90
90 90

/* 3. listing 8 — перевантаження операторів за допомогою дружніх функцій.
Дружні функції не можуть перезавантажити оператори ”=”, ”()”, ”[]”, ”->”.
Дружні функції не отримують неявного вказівника this, тому при перезавантаженні бінарного оператора функція отримує два параметри, унарного – один.
*/

```
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {} // конструктор потрібний для створення тимчасових об'єктів
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  friend loc operator+(loc op1, loc op2); //  дружня функція
  loc operator-(loc op2);
  loc operator=(loc op2);
  loc operator++();
};

// Оператор ”+” перевантажений дружньою функцією
loc operator+(loc op1, loc op2)
{
  loc temp;

  temp.longitude = op1.longitude + op2.longitude;
  temp.latitude = op1.latitude + op2.latitude;

  return temp;
}

// Перевантажений оператор ”-” з використанням неявного вказівника this
loc loc::operator-(loc op2)
```

```
{
  loc temp;

  // notice order of operands
  temp.longitude = longitude - op2.longitude;
  temp.latitude = latitude - op2.latitude;

  return temp;
}

// Перевантаження оператора "="
loc loc::operator=(loc op2)
{
  longitude = op2.longitude;
  latitude = op2.latitude;

  return *this; // i.e., return object that generated call
}
// Перевантаження оператора "++"
loc loc::operator++()
{
  longitude++;
  latitude++;
  return *this;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30);

  ob1 = ob1 + ob2;
  ob1.show();
  return 0;
}
15 50
```

/* 4. listing 9 – використання дружніх функцій для перезаванаження операторів "++", "--". Так як дружні функції не отримують вказівника this, їх операнди потрібно передавати за допомогою посилань
*/

```
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  loc operator=(loc op2);
  friend loc operator++(loc &op);
  friend loc operator--(loc &op);
};

// Перевантажений оператор "=" для loc.
loc loc::operator=(loc op2)
```

```
{
  longitude = op2.longitude;
  latitude = op2.latitude;
  return *this; // повертається об’єкт, який генерував виклик
}

// Оператор ++ перевантажений дружньою функцією
// Використовується передача параметра за допомогою посилання
loc operator++(loc &op)
{
  op.longitude++;
  op.latitude++;
  return op;
}

// Оператор -- перевантажений дружньою функцією
// Використовується передача параметра за допомогою посилання
loc operator--(loc &op)
{
  op.longitude--;
  op.latitude--;
  return op;
}

int main()
{
  loc ob1(10, 20), ob2;

  ob1.show();
  ++ob1;
  ob1.show(); // displays 11 21

  ob2 =  ++ob1;
  ob2.show(); // displays 12 22

  --ob2;
  ob2.show(); // displays 11 21

  return 0;
}
10 20
11 21
12 22
11 21
```

Примітка. Якщо потрібно перезавантажити постфіксу форму операторів ”++”, ”--” то потрібно задати другий фіктивний параметр

```
// перезавантаження постфіксного оператора за допомогою дружньої функції
friend loc operator++(loc &op, int x);
```

/* 5. listing 11 — використання дружніх операторних функцій для додавання об'єкт + число (Ob + 100), число + об'єкт (100 + Ob). При використанні вказівника this вираз 100+Ob невизначений. Використання дружніх функцій усовує цей недолік, так як для них неважливий порядок слідування операндів.

```
*/
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
```

```
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  friend loc operator+(loc op1, int op2);
  friend loc operator+(int op1, loc op2);
};

// Оператор + перевантажений для додавання  об'єкт + int
loc operator+(loc op1, int op2)
{
  loc temp;

  temp.longitude = op1.longitude + op2;
  temp.latitude = op1.latitude + op2;

  return temp;
}
// Оператор + перевантажений для додавання int + об'єкт
loc operator+(int op1, loc op2)
{
  loc temp;

  temp.longitude = op1 + op2.longitude;
  temp.latitude = op1 + op2.latitude;

  return temp;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30), ob3(7, 14);

  ob1.show();
  ob2.show();
  ob3.show();

  ob1 = ob2 + 10; // обидва вирази
  ob3 = 10 + ob2; // вірні

  ob1.show();
  ob3.show();

  return 0;
}
```

10 20
5 30
7 14
15 40
15 40

```
/* 6. listing 13  - перевантаження операторів new, delete для конкретного класу */
#include <iostream>
#include <cstdlib>
#include <new>
using namespace std;

class loc {
  int longitude, latitude;
```

```
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  void *operator new(size_t size);
  void operator delete(void *p);
};

// оператор new перевантажений для класу loc
void *loc::operator new(size_t size)
{
  void *p;

  cout << "Всередені перезавантаженого оператора new\n";
  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
  return p;
}

// оперетор delete перевантажений для класу loc
void loc::operator delete(void *p)
{
  cout << "Всередені перевантаженого оператора delete.\n";
  free(p);
}

int main()
{
  loc *p1, *p2;

  try {
    p1 = new loc (10, 20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні памяті для об'єкта p1.\n";
    return 1;
  }

  try {
    p2 = new loc (-10, -20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні памяті для об'єкта p2.\n";
    return 1;
  }

  p1->show();
  p2->show();

  delete p1;
  delete p2;

  return 0;
}
```

Всередені перезавантаженого оператора new

```
Всередені перезавантаженого оператора new
10 20
-10 -20
Всередені перезавантаженого оператора delete
Всередені перезавантаженого оператора delete
```

```
/* 7. listing 15 — глобальне перевантаження операторів new, delete.
Якщо оператори new, delete перевантажити поза класами, то вони будуть глобальними.
*/
#include <iostream>
#include <cstdlib>
#include <new>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }
};

// глобальний оператор new
void *operator new(size_t size)
{
  void *p;

  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
  return p;
}

// глобальний оператор delete
void operator delete(void *p)
{
  free(p);
}

int main()
{
  loc *p1, *p2;
  float *f;

  try {
    p1 = new loc (10, 20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні пам'яті для об'єкта p1.\n";
    return 1;
  }

  try {
    p2 = new loc (-10, -20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні пам'яті для об'єкта p2.\n";
```

```
    return 1;
  }
 try {
    f = new float; // використовується перезавантажена версія оператора new
  } catch (bad_alloc xa) {
    cout << "Помилка виділення пам'яті.\n";
    return 1;
  }

  *f = 10.10F;
  cout << *f << "\n";

  p1->show();
  p2->show();

  delete p1;
  delete p2;
  delete f;

  return 0;
}
```

10.1
10 20
-10 -20

/* 8. listing 17 – перевантаження операторів new, delete для масивів.
Розміщення і звільнення з памяті масиву об’єктів loc */

```
#include <iostream>
#include <cstdlib>
#include <new>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {longitude = latitude = 0;}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  void *operator new(size_t size);
  void operator delete(void *p);

  void *operator new[](size_t size);
  void operator delete[](void *p);
};

// перевантажений оператор new для класу loc.
void *loc::operator new(size_t size)
{
void *p;

  cout << "In overloaded new.\n";
  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
```

```
  return p;
}

// перевантажений оператор delete для класу loc.
void loc::operator delete(void *p)
{
  cout << "In overloaded delete.\n";
  free(p);
}

// перевантажений оператор new для масиву об’єктів.
void *loc::operator new[](size_t size)
{
  void *p;

  cout << "Using overload new[].\n";
  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
  return p;
}

// перевантажений оператор delete для масиву об’єктів loc.
void loc::operator delete[](void *p)
{
  cout << "Freeing array using overloaded delete[]\n";
  free(p);
}

int main()
{
  loc *p1, *p2;
  int i;

  try {
    p1 = new loc (10, 20); // розміщення об’єкту в пам’яті
  } catch (bad_alloc xa) {
    cout << "Allocation error for p1.\n";
    return 1;;
  }

  try {
    p2 = new loc [10]; // розміщення масиву в пам’яті
  } catch (bad_alloc xa) {
    cout << "Allocation error for p2.\n";
    return 1;;
  }

  p1->show();

  for(i=0; i<10; i++)
    p2[i].show();

  delete p1; // знищення об'єкту
  delete [] p2; // знищення масиву

  return 0;
}
```

/* 9. listing 21 — перевантаження оператора []. При перезавантаженні оператор [] вважається бінарним, тому він має мати такий вид: operator[](int i) */

```
#include <iostream>
```

```
using namespace std;

class atype {
  int a[3];
public:
  atype(int i, int j, int k) {
    a[0] = i;
    a[1] = j;
    a[2] = k;
  }
  int operator[](int i) { return a[i]; }
};

int main()
{
  atype ob(1, 2, 3);
  cout << ob[1]; // виводить число 2

return 0;
}
```
2

/* 10. listing 22 перевантаження оператора []. Для того, щоб оператор [] міг бути і в лівій і правій частині функція operator[] має повертати посилання */

```
#include <iostream>
using namespace std;

class atype {
  int a[3];
public:
  atype(int i, int j, int k) {
    a[0] = i;
    a[1] = j;
    a[2] = k;
  }
  int &operator[](int i) { return a[i]; }
};

int main()
{
  atype ob(1, 2, 3);
  cout << ob[1]; // Оператор [] стоїть справа, виводить на екран 2
  cout << " ";
  ob[1] = 25;    // Оператор [] стоїть зліва від ”=”.
  cout << ob[1]; // виводить на екран 25
  return 0;
}
```
2 25

/* 11. listing 23 — використання перевантаженого оператора [] для перевірки діапазону індексів масиву */

```
// Приклад безпечного масиву
#include <iostream>
#include <cstdlib>
using namespace std;

class atype {
  int a[3];
public:
  atype(int i, int j, int k) {
    a[0] = i;
    a[1] = j;
    a[2] = k;
  }
```

```
  int &operator[](int i);
};

// Перевірка діапазону всередині класу atype.
int &atype::operator[](int i)
{
  if(i<0 || i> 2) {
    cout << " Вихід за допустимий діапазон\n";
    exit(1);
  }
  return a[i];
}

int main()
{
  atype ob(1, 2, 3);

  cout << ob[1]; // виводить на екран  2
  cout << " ";

  ob[1] = 25;    // [] оператор зліва
  cout << ob[1]; // виводить на екран  25

  ob[3] = 44;    // Генерується помилка, значення 3 лежить за межами допустимого
інтервалу

  return 0;
}
```

2 25 Вихід за допустимий діапазон

/* 12. listing 28 — перевантаження оператора (). При перезавантаженні оператора ”()” створюється операторна функція, якій можна передавати довільне число параметрів */

```
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  loc operator+(loc op2);
  loc operator()(int i, int j);
};

// Перевантажений оператор ”()” для класу loc
loc loc::operator()(int i, int j)
{
  longitude = i;
  latitude = j;

  return *this;
}

// Перевантажений оператор ”+” для класу loc
```

```
loc loc::operator+(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}

int main()
{
  loc ob1(10, 20), ob2(1, 1);
  ob1.show();
  ob1(7, 8);  // Оператор () застосовується самостійно
  ob1.show();
  ob1 = ob2 + ob1(10, 10);  // оператор () використовується у виразі
  ob1.show();
  return 0;
}
```

10 20
7 8
11 11

/* 13. listing 29 — оператор ”->” при перевантаженні вважається унарним */

```
#include <iostream>
using namespace std;

class myclass {
public:
 int i;
 myclass *operator->() {return this;} //операторна функція operator->() має бути членом класу
};

int main()
{
  myclass ob;
  ob->i = 10; // еквівалентно виразу ob.i
  cout << ob.i << " " << ob->i;
  return 0;
}
```

10 10

/* 14. listing 30 — перевантаження оператора ”,”. Оператор вважається бінарним. Мета перезавантаження може бути різна. В даному випадку будуть ігноруватися всі аргументи, записані через кому, крім крайнього правого */

```
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }
```

```
  loc operator+(loc op2);
  loc operator,(loc op2);
};

// Перевантаження оператора ”,” для класу loc
loc loc::operator,(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude;
  temp.latitude = op2.latitude;
  cout << op2.longitude << " " << op2.latitude << "\n";

  return temp;
}

// Перезавантаження оператора ”+” для класу loc
loc loc::operator+(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}
int main()
{
  loc ob1(10, 20), ob2( 5, 30), ob3(1, 1);
  ob1.show();
  ob2.show();
  ob3.show();
  cout << "\n";
  ob1 = (ob1, ob2+ob2, ob3);
  ob1.show(); // виводиться на екран значення 1 1, тобто значення ob3
  return 0;
}
```

```
10 20
5 30
1 1

10 60
1 1
1 1
```